Министерство науки и высшего образования Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт перспективной инженерии
Департамент цифровых, робототехнических систем и электроники

ОТЧЕТ
ПО ЛАБОРАТОРНОЙ РАБОТЕ №3
дисциплины
«Объектно-ориентированное программирование»
Вариант № 18

Выполнил:
Текеева Мадина Азрет-Алиевна
3 курс, группа ИВТ-б-о-23-2,
09.03.01 «Информатика и вычислительная техника»,
направленность (профиль) «Программное обеспечение средств вычислительной техники и автоматизированных систем», очная форма обучения

_____
(подпись)

Проверил:
Доцент департамента цифровых, робототехнических систем и электроники института перспективной инженерии Воронкин Р.А

_____
(подпись)

Отчет защищен с оценкой _____ Дата защиты_____

Ставрополь, 2025 г.

Тема: Наследование и полиморфизм в языке Python.

Цель работы: приобретение навыков по созданию иерархии классов при написании программ с помощью языка программирования Python версии 3.x.

Репозиторий: https://github.com/bickrosss/TM_oop_lab3

Порядок выполнения работы:

1. Пример использования абстрактного класса и абстрактного метода.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from abc import ABC, abstractmethod


class Polygon(ABC):

    @abstractmethod
    def noofsides(self) -> None:
        pass


class Triangle(Polygon):

    def noofsides(self) -> None:
        print("I have 3 sides")


class Pentagon(Polygon):

    def noofsides(self) -> None:
        print("I have 5 sides")


class Hexagon(Polygon):

    def noofsides(self) -> None:
        print("I have 6 sides")


class Quadrilateral(Polygon):

    def noofsides(self) -> None:
        print("I have 4 sides")
```

```
R = Triangle()
R.noofsides()

K = Quadrilateral()
K.noofsides()

P = Pentagon()
P.noofsides()

H = Hexagon()
H.noofsides()
```

```
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
```

Рисунок 1. Работа примера

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from abc import ABC, abstractmethod


class Animal(ABC):
    @abstractmethod
    def move(self) -> None:
        pass


class Human(Animal):

    def move(self) -> None:
        print("I can walk and run")


class Snake(Animal):

    def move(self) -> None:
        print("I can crawl")


class Dog(Animal):

    def move(self) -> None:
        print("I can bark")
```

```
class Lion(Animal):

    def move(self) -> None:
        print("I can roar")


H = Human()
H.move()

S = Snake()
S.move()

D = Dog()
D.move()

K = Lion()
K.move()
```

```
I can walk and run
I can crawl
I can bark
I can roar
```

Рисунок 2. Работа примера

2. Пример использования абстрактного класса с абстрактными свойствами.

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-


from abc import ABC, abstractmethod


class parent(ABC):

    @property
    @abstractmethod
    def geeks(self) -> str:
        pass


class child(parent):

    @property
    def geeks(self) -> str:
        return "child class"
```

```
try:
    r = parent()  # type: ignore[abstract]
    print(r.geeks)
except Exception as err:
    print(err)

r = child()
print(r.geeks)
```

```
Can't instantiate abstract class parent without an implementation for
abstract method 'geeks'
child class
```

Рисунок 3. Вывод данных

3. Выполнили задание 1.

Создать класс Pair (пара целых чисел); определить методы изменения полей и операцию сложения пар (a, b) + (c,d) = (a+b,c +d). Определить класс-наследник Long с полями: старшая часть числа и младшая часть числа. Переопределить операцию сложения и определить методы умножения и вычитания.

Листинг программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from math import gcd

class Pair:
    def __init__(self, first=0, second=0):
        self.__first = int(first)
        self.__second = int(second)

    @property
    def first(self):
        return self.__first

    @first.setter
    def first(self, value):
        self.__first = int(value)

    @property
    def second(self):
        return self.__second

    @second.setter
```

```
    def second(self, value):
        self.__second = int(value)

    def read(self):
        self.__first = int(input("Введите первое число: "))
        self.__second = int(input("Введите второе число: "))

    def display(self):
        print(f"({self.__first}, {self.__second})")

    def add(self, other):
        if isinstance(other, Pair):
            return Pair(self.__first + other.__first, self.__second +
other.__second)
        else:
            raise ValueError("Операнд должен быть типа Pair")

    def __add__(self, other):
        return self.add(other)

    def __str__(self):
        return f"({self.__first}, {self.__second})"


class Long(Pair):
    def __init__(self, high=0, low=0):
        super().__init__(high, low)

    @property
    def high(self):
        return self.first

    @high.setter
    def high(self, value):
        self.first = value

    @property
    def low(self):
        return self.second

    @low.setter
    def low(self, value):
        self.second = value

    def read(self):
        self.high = int(input("Введите старшую часть числа: "))
        self.low = int(input("Введите младшую часть числа: "))

    def display(self):
        print(f"Старшая часть={self.high}, младшая часть={self.low}")
```

```
    def add(self, other):
        if isinstance(other, Long):
            new_high = self.high + other.high
            new_low = self.low + other.low

            # Обработка переполнения младшей части
            if new_low >= 1000:
                new_high += new_low // 1000
                new_low = new_low % 1000

            return Long(new_high, new_low)
        else:
            raise ValueError("Операнд должен быть типа Long")

    def multiply(self, other):
        if isinstance(other, Long):
            # Упрощенное умножение (для демонстрации)
            total_high = self.high * other.high
            total_low = self.low * other.low

            # Нормализация
            if total_low >= 1000:
                total_high += total_low // 1000
                total_low = total_low % 1000

            return Long(total_high, total_low)
        else:
            raise ValueError("Операнд должен быть типа Long")

    def subtract(self, other):
        if isinstance(other, Long):
            new_high = self.high - other.high
            new_low = self.low - other.low

            # Обработка отрицательной младшей части
            if new_low < 0:
                new_high -= 1
                new_low += 1000

            return Long(new_high, new_low)
        else:
            raise ValueError("Операнд должен быть типа Long")

    def __mul__(self, other):
        return self.multiply(other)

    def __sub__(self, other):
        return self.subtract(other)


if __name__ == '__main__':
```

```
print("Демонстрация класса Pair")
p1 = Pair(10, 20)
p2 = Pair(5, 15)

print("Пары чисел:")
print(f"p1 = ", end="")
p1.display()
print(f"p2 = ", end="")
p2.display()

result_pair = p1 + p2
print("Сложение пар: ", end="")
result_pair.display()

print("\nДемонстрация класса Long")
l1 = Long(2, 500)
l2 = Long(1, 600)

print("Long числа:")
l1.display()
l2.display()

result_add = l1 + l2
print("Сложение Long: ", end="")
result_add.display()

result_mult = l1 * l2
print("Умножение Long: ", end="")
result_mult.display()

result_sub = l1 - l2
print("Вычитание Long: ", end="")
result_sub.display()

print("\nТестирование переполнения")
l3 = Long(1, 800)
l4 = Long(1, 300)

print("Тест переполнения:")
l3.display()
l4.display()

result_overflow = l3 + l4
print("Результат (с обработкой переполнения): ", end="")
result_overflow.display()

print("Создадим новую пару через ввод:")
p3 = Pair()
p3.read()
print("Введенная пара: ", end="")
p3.display()
```

```
    print("Создадим новое Long число через ввод:")
    l5 = Long()
    l5.read()
    print("Введенное число: ", end="")
    l5.display()
```

Пример использования программы:

```
from pair import Pair, Long


def main():

    # Демонстрация базового класса Pair
    print("1. Демонстрация класса pair")
    print("-" * 40)

    # Создание пар
    pair1 = Pair(15, 25)
    pair2 = Pair(7, 13)

    print("Созданы пары чисел:")
    print(f"Пара 1: {pair1}")
    print(f"Пара 2: {pair2}")

    # Операции с парами
    sum_pair = pair1 + pair2
    print(f"\nРезультат сложения пар: {sum_pair}")

    # Изменение полей
    print("\nИзменение полей пары 1:")
    pair1.first = 100
    pair1.second = 200
    print(f"После изменения: {pair1}")

    # Демонстрация класса-наследника Long
    print("\n2. Демонстрация класса Long")
    print("-" * 40)

    # Создание Long чисел
    long1 = Long(3, 750)
    long2 = Long(2, 450)

    print("Созданы Long числа:")
    long1.display()
    long2.display()

    # Арифметические операции с Long числами
    print("\nАрифметические операции:")
```

```
    sum_long = long1 + long2
    print("Сложение: ", end="")
    sum_long.display()

    mult_long = long1 * long2
    print("Умножение: ", end="")
    mult_long.display()

    sub_long = long1 - long2
    print("Вычитание: ", end="")
    sub_long.display()

    # Демонстрация обработки переполнения
    print("\n3. Демонстрация обработки переполнения")
    print("-" * 40)

    long3 = Long(1, 800)
    long4 = Long(1, 500)

    print("Числа для теста переполнения:")
    long3.display()
    long4.display()

    result = long3 + long4
    print("Результат сложения (с обработкой переполнения): ", end="")
    result.display()

    # Работа с пользовательским вводом
    print("\n4. Работа с пользовательским вводом")
    print("-" * 40)

    print("Создание пары через ввод:")
    user_pair = Pair()
    user_pair.read()
    print(f"Вы ввели пару: {user_pair}")

    print("\nСоздание Long числа через ввод:")
    user_long = Long()
    user_long.read()
    print("Вы ввели: ", end="")
    user_long.display()

if __name__ == '__main__':
    main()
```

```
1. Демонстрация класса pair
----------------------------------------
Созданы пары чисел:
Пара 1: (15, 25)
Пара 2: (7, 13)

Результат сложения пар: (22, 38)

Изменение полей пары 1:
После изменения: (100, 200)

2. Демонстрация класса Long
----------------------------------------
Созданы Long числа:
Старшая часть=3, младшая часть=750
Старшая часть=2, младшая часть=450

Арифметические операции:
Сложение: Старшая часть=6, младшая часть=200
Умножение: Старшая часть=343, младшая часть=500
Вычитание: Старшая часть=1, младшая часть=300

3. Демонстрация обработки переполнения
----------------------------------------
Числа для теста переполнения:
Старшая часть=1, младшая часть=800
Старшая часть=1, младшая часть=500
Результат сложения (с обработкой переполнения): Старшая часть=3, младшая часть=300

4. Работа с пользовательским вводом
----------------------------------------
Создание пары через ввод:
Введите первое число: 5
Введите второе число: 6
Вы ввели пару: (5, 6)
```

Рисунок 4. Результат

4. Решили задание 2.

В следующих заданиях требуется реализовать абстрактный базовый класс, определив в нем абстрактные методы и свойства. Эти методы определяются в производных классах. Для ввода использовать вспомогательный класс Reader, позволяющий читать данные с клавиатуры. Для вывода использовать методы __str__ или __repr__.

Вызывающая программа должна продемонстрировать все варианты вызова переопределенных абстрактных методов. Написать функцию вывода,

получающую параметры базового класса по ссылке и демонстрирующую виртуальный вызов.

Создать абстрактный базовый класс Figure с абстрактными методами вычисления площади и периметра. Создать производные классы: Rectangle (прямоугольник), Circle (круг), Trapezium (трапеция) со своими функциями площади и периметра. Самостоятельно определить, какие поля необходимы, какие из них можно задать в базовом классе, а какие - в производных. Площадь трапеции: S=(a+b)*h/2.

Листинг программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from abc import ABC, abstractmethod
from math import pi, sqrt

class Reader:
    @staticmethod
    def read_float(prompt):
        while True:
            try:
                return float(input(prompt))
            except ValueError:
                print("Ошибка: введите число")

    @staticmethod
    def read_int(prompt):
        while True:
            try:
                return int(input(prompt))
            except ValueError:
                print("Ошибка: введите целое число")

class Figure(ABC):
    @abstractmethod
    def area(self):
        pass

    @abstractmethod
    def perimeter(self):
        pass

    @abstractmethod
    def is_valid(self):
        pass
```

```
    def __str__(self):
        return f"{self.__class__.__name__}: площадь = {self.area():.2f}, периметр
= {self.perimeter():.2f}"

class Rectangle(Figure):
    def __init__(self, width, height):
        self.width = width
        self.height = height
        if not self.is_valid():
            raise ValueError("Некорректные параметры прямоугольника")

    def is_valid(self):
        return self.width > 0 and self.height > 0

    def area(self):
        return self.width * self.height

    def perimeter(self):
        return 2 * (self.width + self.height)

class Circle(Figure):
    def __init__(self, radius):
        self.radius = radius
        if not self.is_valid():
            raise ValueError("Некорректный параметр круга")

    def is_valid(self):
        return self.radius > 0

    def area(self):
        return pi * self.radius ** 2

    def perimeter(self):
        return 2 * pi * self.radius

class Trapezium(Figure):
    def __init__(self, base1, base2, height):
        self.base1 = base1
        self.base2 = base2
        self.height = height
        # Вычисляем боковые стороны (для упрощения считаем равнобедренной)
        self.side = sqrt(((base1 - base2) / 2) ** 2 + height ** 2)
        if not self.is_valid():
            raise ValueError("Некорректные параметры трапеции")

    def is_valid(self):
        # Все параметры положительные и основания разные
        return (self.base1 > 0 and self.base2 > 0 and
                self.height > 0 and self.base1 != self.base2)
```

```
    def area(self):
        return (self.base1 + self.base2) * self.height / 2

    def perimeter(self):
        return self.base1 + self.base2 + 2 * self.side

def demonstrate_virtual_call(figure: Figure):
    """Функция, демонстрирующая виртуальный вызов"""
    print(f"Виртуальный вызов для {figure.__class__.__name__}:")
    print(f"  Площадь: {figure.area():.2f}")
    print(f"  Периметр: {figure.perimeter():.2f}")
    print(f"  Корректность: {'Да' if figure.is_valid() else 'Нет'}")
    print()
```

Пример использования программы:

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

from figure_package.figure import Figure, Rectangle, Circle, Trapezium, Reader,
demonstrate_virtual_call

def create_figures_with_reader():
    """Создание фигур с использованием класса Reader"""
    reader = Reader()
    figures = []

    print("Создание прямоугольника:")
    width = reader.read_float("Введите ширину: ")
    height = reader.read_float("Введите высоту: ")
    figures.append(Rectangle(width, height))

    print("\nСоздание круга:")
    radius = reader.read_float("Введите радиус: ")
    figures.append(Circle(radius))

    print("\nСоздание трапеции:")
    base1 = reader.read_float("Введите первое основание: ")
    base2 = reader.read_float("Введите второе основание: ")
    height = reader.read_float("Введите высоту: ")
    figures.append(Trapezium(base1, base2, height))

    return figures

def main():
    print("Демонстарция наследования и полиморфизма\n")

    # Создание фигур
    figures = create_figures_with_reader()

    print("\n" + "="*50)
```

```
    print("Демонстрация виртуальных вызовов:")
    print("="*50)

    # Демонстрация полиморфизма
    for figure in figures:
        demonstrate_virtual_call(figure)
        print(f"Вызов __str__: {figure}")
        print("-" * 30)

if __name__ == '__main__':
    main()
```

```
Демонстрация наследования и полиморфизма

Создание прямоугольника:
Введите ширину: 7
Введите высоту: 4

Создание круга:
Введите радиус: 6

Создание трапеции:
Введите первое основание:
Ошибка: введите число
Введите первое основание: 7
Введите второе основание: 3
Введите высоту: 7
Демонстрация виртуальных вызовов:
Виртуальный вызов для Rectangle:
  Площадь: 28.00
  Периметр: 22.00
  Корректность: Да

Вызов __str__: Rectangle: площадь = 28.00, периметр = 22.00
Виртуальный вызов для Circle:
  Площадь: 113.10
  Периметр: 37.70
  Корректность: Да

Вызов __str__: Circle: площадь = 113.10, периметр = 37.70
Виртуальный вызов для Trapezium:
  Площадь: 35.00
  Периметр: 24.56
  Корректность: Да

Вызов __str__: Trapezium: площадь = 35.00, периметр = 24.56
```

Рисунок 5. Результат

Вывод: в ходе выполнения лабораторной работы были приобретены навыки по созданию иерархии классов при написании программ с помощью языка программирования Python.